FREE eBook

LEARNING lucene

#lucene

Table of Contents

About

Chapter 1: Getting started with lucene

Remarks

Apache Lucene is a Java-based full text search library.

Versions

Version	Release Date
2.9.4	2010-12-03
3.0.3	2010-12-03
3.6.2	2013-01-16
4.10.4	2015-10-14
5.5.2	2016-06-24
6.3.0	2016-11-08

Examples

Setup

Lucene is a Java library. If you don't have a Java development environment set up already, see the Java documentation.

Download the latest version of Lucene from the Apache website, and unzip it.

Add the required jars to your classpath. The following jars will be required by many projects, including the Hello World example here:

- `core/lucene-core-6.1.0.jar`: Core Lucene functionality.
- `core/analysis/common/lucene-analyzers-common-6.1.0.jar`: Provides a variety of analyzers, including the ubiquitous StandardAnalyzer.
- `queryparser/lucene-queryparser-6.1.0.jar`: Provides the query parser.

Place the code in `HelloLucene.java`. Compile it with this command:

```
javac -classpath "core/*:queryparser/*" HelloLucene.java
```

And run it with this command:

```
java -classpath ".:core/*:queryparser/*" HelloLucene
```

Hello World

This basic Lucene example creates a simple index, and searches on it.

Note: RAMDirectory creates a memory-resident index, and is handy for experimenting and testing, but in practice most people will need to have an index stored in the file system (see FSDirectory.open).

```
import java.io.IOException;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.*;
import org.apache.lucene.index.*;
import org.apache.lucene.queryparser.classic.*;
import org.apache.lucene.search.*;
import org.apache.lucene.store.*;

public class HelloLucene {
    public static void main(String[] args) throws IOException, ParseException
    {
        //Create a new index and open a writer
        Directory dir = new RAMDirectory();
        Analyzer analyzer = new StandardAnalyzer();
        IndexWriterConfig config = new IndexWriterConfig(analyzer);
        IndexWriter writer = new IndexWriter(dir, config);

        //Create a document to index
        Document doc = new Document();
        doc.add(new TextField("text", "Hello World!", Field.Store.YES));

        //Index the document and close the writer
        System.out.println("Indexing document: " + doc);
        writer.addDocument(doc);
        writer.close();

        //Open an IndexSearcher
        IndexReader reader = DirectoryReader.open(dir);
        IndexSearcher searcher = new IndexSearcher(reader);

        //Create a query
        QueryParser parser = new QueryParser("text", analyzer);
        Query query = parser.parse("world");

        //Search for results of the query in the index
        System.out.println("Searching for: \"" + query + "\"");
        TopDocs results = searcher.search(query, 10);
        for (ScoreDoc result : results.scoreDocs) {
            Document resultDoc = searcher.doc(result.doc);
            System.out.println("score: " + result.score +
                    " -- text: " + resultDoc.get("text"));
        }
        reader.close();
    }
}
```

Read Getting started with lucene online: https://riptutorial.com/lucene/topic/3132/getting-started-with-lucene

Chapter 2: Analysis

Examples

Creating a custom analyzer

Most analysis customization is in the `createComponents` class, where the Tokenizer and TokenFilters are defined.

CharFilters can be added in the `initReader` method.

```
Analyzer analyzer = new Analyzer() {
    @Override
    protected Reader initReader(String fieldName, Reader reader) {
        return new HTMLStripCharFilter(reader);
    }

    @Override
    protected TokenStreamComponents createComponents(String fieldName) {
        Tokenizer tokenizer = new StandardTokenizer();
        TokenStream stream = new StandardFilter(tokenizer);
        //Order matters!  If LowerCaseFilter and StopFilter were swapped here, StopFilter's
        //matching would be case sensitive, so "the" would be eliminated, but not "The"
        stream = new LowerCaseFilter(stream);
        stream = new StopFilter(stream, StopAnalyzer.ENGLISH_STOP_WORDS_SET);
        return new TokenStreamComponents(tokenizer, stream);
    }
};
```

Iterating manually through analyzed tokens

```
TokenStream stream = myAnalyzer.tokenStream("myField", textToAnalyze);
stream.addAttribute(CharTermAttribute.class);
stream.reset();
while(stream.incrementToken()) {
    CharTermAttribute token = stream.getAttribute(CharTermAttribute.class);
    System.out.println(token.toString());
}

stream.close();
```

A number of Attributes are available. The most common is `CharTermAttribute`, which is used to get the analyzed term as a String.

Read Analysis online: https://riptutorial.com/lucene/topic/4830/analysis

Chapter 3: Deleting Documents using a Multi-Term Query

Introduction

Deleting documents from a Lucene index is easy when you have a primary key field in your document (like in traditional SQL databases).

However, sometimes deleting a number of documents based on multiple fields in the document is what you need. The Lucene API allows you to achieve this by specifying a query to use for deletion.

To do this, pick the right Analyzer, construct the query, pass the query to the indexWriter to delete the documents.

Syntax

1. indexWriter.deleteDocuments(multiTermQuery);
2. Query multiTermQuery = new QueryParser("", analyzer).parse("field_name1:"field value 1" AND field_name2:"field value 2"");
3. BooleanQuery multiTermQuery = new BooleanQuery(); multiTermQuery.add(new TermQuery(new Term("field_name1", "field value 1")), BooleanClause.Occur.MUST); multiTermQuery.add(new TermQuery(new Term("field_name2", "field value 2")), BooleanClause.Occur.MUST);

Remarks

Caveats with the Choice of Analyzer

It's not immediately obvious, but the analyzer that you are using makes a huge difference to the way your query is run. This is because the StandardAnalyzer filters out common English words like "the" and "a". You might want to pick a different analyzer (like KeywordAnalyzer) so that it matches exactly. This obviously depends on you application of Lucene of course.

Examples

Choice of Analyzer

First of all, watch out which analyzer you are using. I was stumped for a while only to realise that the StandardAnalyzer filters out common words like 'the' and 'a'. This is a problem when your field has the value 'A'. You might want to consider the KeywordAnalyzer:

See this post around the analyzer.

```
// Create an analyzer:
// NOTE: We want the keyword analyzer so that it doesn't strip or alter any terms:
// In our example, the Standard Analyzer removes the term 'A' because it is a common English
word.
// http://stackoverflow.com/a/9071806/231860
KeywordAnalyzer analyzer = new KeywordAnalyzer();
```

Query Parser

Next, you can either create your query using the QueryParser:

See this post around overriding the default operator.

```
// Create a query parser without a default field in this example (the first argument):
QueryParser queryParser = new QueryParser("", analyzer);

// Optionally, set the default operator to be AND (we leave it the default OR):
// http://stackoverflow.com/a/9084178/231860
// queryParser.setDefaultOperator(QueryParser.Operator.AND);

// Parse the query:
Query multiTermQuery = queryParser.parse("field_name1:\"field value 1\" AND
field_name2:\"field value 2\"");
```

Query API

Or you can achieve the same by constructing the query yourself using their API:

See this tutorial around creating the BooleanQuery.

```
BooleanQuery multiTermQuery = new BooleanQuery();
multiTermQuery.add(new TermQuery(new Term("field_name1", "field value 1")),
BooleanClause.Occur.MUST);
multiTermQuery.add(new TermQuery(new Term("field_name2", "field value 2")),
BooleanClause.Occur.MUST);
```

Delete the Documents that Match the Query

Then we finally pass the query to the writer to delete documents that match the query:

See the answer to this question.

See the API here

```
// Remove the document by using a multi key query:
// http://www.avajava.com/tutorials/lessons/how-do-i-combine-queries-with-a-boolean-query.html
indexWriter.deleteDocuments(multiTermQuery);
```

Read Deleting Documents using a Multi-Term Query online:
https://riptutorial.com/lucene/topic/9959/deleting-documents-using-a-multi-term-query

Chapter 4: Queries

Examples

BooleanQuery

BooleanQuery is used to combine other queries.

They can be combined using three BooleanClause.Occur parameters:

- `BooleanClause.Occur.MUST` - The subquery must be matched.
- `BooleanClause.Occur.SHOULD` - The subquery may not be matched, but will be scored more highly if it is. If there are no MUST clauses, then at least one SHOULD clause must be matched.
- `BooleanClause.Occur.MUST_NOT` - The subquery must not match the document.

In this example, a document will match if it has "important", but *not* "forbidden", and will get a higher score if it also has "helpful".

```
Query importantQuery = new TermQuery(new Term("fieldname", "important"));
Query optionalQuery = new TermQuery(new Term("fieldname", "helpful"));
Query forbidQuery = new TermQuery(new Term("fieldname", "forbidden"));
BooleanQuery query = new BooleanQuery.Builder()
        .add(importantQuery, BooleanClause.Occur.MUST)
        .add(optionalQuery, BooleanClause.Occur.SHOULD)
        .add(forbidQuery, BooleanClause.Occur.MUST_NOT)
        .build();
```

Alternatively, you can also specify the minimum number of clauses that must be matched:

```
Query query1 = new TermQuery(new Term("fieldname", "one"));
Query query2 = new TermQuery(new Term("fieldname", "two"));
Query query3 = new TermQuery(new Term("fieldname", "three"));
BooleanQuery query = new BooleanQuery.Builder()
        .add(query1, BooleanClause.Occur.SHOULD)
        .add(query2, BooleanClause.Occur.SHOULD)
        .add(query3, BooleanClause.Occur.SHOULD)
        .setMinimumNumberShouldMatch(2)
        .build();
```

Gotcha: Clauses with `BooleanClause.Occur.MUST_NOT` *do not* match everything else, they only eliminate matches. Your BooleanQuery must have at least one `MUST` or `SHOULD` clause, or it will match nothing. This, for example, will **NOT** work:

```
//***This does NOT work!***
Query forbidQuery = new TermQuery(new Term("fieldname", "forbidden"));
BooleanQuery getEverythingElseQuery = new BooleanQuery.Builder()
        .add(forbidQuery, BooleanClause.Occur.MUST_NOT)
        .build();
```

PhraseQuery

PhraseQuery is used to search for a sequence of terms. The following matches the phrase "Hello World" (after being indexed with `StandardAnalyzer`)

```
Query query = new PhraseQuery.Builder()
        .add(new Term("text", "hello"))
        .add(new Term("text", "world"))
        .build();
```

PhraseQuery can also handle "slop", or extra terms within a query, by setting a maximum edit distance with `setSlop`. This will match "Lorem ipsum sit amet dolor":

```
Query query = new PhraseQuery.Builder()
        .add(new Term("text", "lorem"))
        .add(new Term("text", "amet"))
        .setSlop(2)
        .build();
```

You can also set exact position increments:

```
Query query = new PhraseQuery.Builder()
        .add(new Term("text", "lorem"), 0)
        .add(new Term("text", "sit"), 2)
        .add(new Term("text", "dolor"), 4)
        .build();
```

DisjunctionMaxQuery

This combines queries such that the best (that is, highest-scoring) match of it's subqueries contributes to the final score.

```
List<Query> disjuncts = new ArrayList<Query>();
disjuncts.add(new TermQuery(new Term("fieldname", "hello")));
disjuncts.add(new TermQuery(new Term("fieldname", "world")));
Query query = new DisjunctionMaxQuery(disjuncts, 0.0f);
```

The second argument to the `DisjunctionMaxQuery` constructor is a tiebreaker value, which, when non-zero, allows non-maximal matches to make some small contribution to score, in order to break ties. It should generally be small (on the order of 0.1).

Boosting queries

A query can be boosted to increase it's score relative to other subqueries. This is done by wrapping it with a BoostQuery

```
Query lessRelevantQuery = new TermQuery(new Term("fieldname", "ipsum"));
//Five times as interesting
Query highlyRelevantQuery = new BoostQuery(
         new TermQuery(new Term("fieldname", "lorem")),
         5.0f);
BooleanQuery query = new BooleanQuery.Builder()
        .add(lessRelevantQuery, BooleanClause.Occur.SHOULD)
        .add(highlyRelevantQuery, BooleanClause.Occur.SHOULD)
        .build();
```

Read Queries online: https://riptutorial.com/lucene/topic/5614/queries

Credits

S. No	Chapters	Contributors
1	Getting started with lucene	Community, femtoRgon, Hamid Rohani, Heidar
2	Analysis	femtoRgon
3	Deleting Documents using a Multi-Term Query	Luke Machowski
4	Queries	femtoRgon